Universitatea din Bucureşti
Facultatea de Matematică şi Informatică
Departamentul de Informatică

Iuliana Georgescu
Bogdan Alexe
Radu Ionescu
Informatică Opţional, anul 3

Vedere Artificială - Tema 3

Detectare facială folosind metoda glisării ferestrei şi histograme de gradienţi orientaţi

Obiectiv:

Scopul acestui proiect este implementarea şi testarea unui algoritm de detectare facială folosind metoda glisării unei ferestre şi histograme de gradienţi orientaţi.

Funcţiile Python care vă vor ajuta la implementarea proiectului sunt în directorul *cod*; imaginile pe care le veţi folosi sunt în directorul *data*.

Introducere. Detectarea obiectelor în imagini este una dintre probleme fundamentale în Vederea Artificială. Detectarea facială înseamnă localizarea tuturor feţelor umane ce apar într-o imagine test (Figura 1). De obicei, localizarea se realizează la nivel de fereastră dreptunghiulară (marcate cu roşu în Figura 1). Camerele digitale moderne precum şi instrumentele de organizare a colecţiilor de fotografii (Picasa, iPhoto) au un asemenea detector facial încorporat.

Una din abordările cu cel mai mare succes în rezolvarea acestei probleme o constituie glisarea unei ferestre (termenul în engleză este *sliding-window*) într-o imagine test şi folosirea unui clasificator care decide pentru fiecare fereastă pe baza pixelilor ei dacă aceasta conţine sau nu o faţă. Veţi urma această paradigmă în cadrul acestei teme.

Implementare. Există multe implementări posibile ale unui detector facial. Primele abordări de mare succes pe tema detectării faciale datează de la lucrările lui Rowley et. al 1998 şi Viola şi Jones 2001. În cadrul acestui proiect veţi implementa un detector facial care foloseşte pentru descrierea conţinutului vizual al fiecărei ferestre dintr-o imagine drept ca-

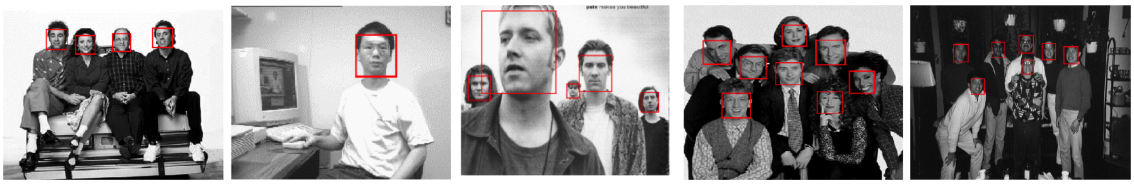

Figura 1: **Exemple de detectare facială**: în fiecare dintre cele 5 imagine sunt localizate la nivel de fereastră dreptunghiulară de culoare roşie toate feţele oamenilor ce apar în imaginea respectivă.

racteristici histograme de gradienţi orientaţi (HOG). Histogramele de gradienţi orientaţi au fost introduse de Dalal şi Triggs în anul 2005 (aveţi articolul ataşat în arhiva cu materiale) şi folosite iniţial pentru detectarea oamenilor. În acest proiect veţi folosi histogramele de gradienţi orientaţi pentru detectarea facială.

Construcţia detectorului facial presupune următoarele etape:

- etapa de antrenare (învăţare): în această etapă se antrenează un clasificator care poate distinge între ferestrele ce conţin feţe de cele ce nu conţin feţe. Veţi folosi acest clasificator în următoarea etapă pentru a clasifica miile de ferestre dreptunghiulare dintr-o imagine test.
- etapa de testare: pentru o imagine test vreţi să localizaţi toate feţele care apar în ea. Veţi realiza acest lucru prin glisarea unei ferestre de la stânga la dreapta şi de sus în jos şi asignarea unui scor fiecărei ferestre pe baza clasificatorului învăţat în etapa anterioară. Maximele locale localizează feţele în imagine. Trebuie să aveţi în vedere că feţele ce apar într-o imagine pot avea mărimi diferite.

Scriptul Python *run_project.py* conţine implementarea întregului proiect, apelând rând pe rând funcţii care realizează paşii de mai sus. Această funcţie este completată în întregime. Sarcina voastră este de a completa bucăţile lipsă din cod astfel încât *run_project.py* să ruleze întregul proiect. Pe baza acestui lucru veţi putea realiza experimentele care vă vor ajuta să răspundeţi întrebărilor de la final.

1.1 Etapa de antrenare (4 puncte)

Exemple de antrenare pozitive şi negative. Antrenarea unui clasificator se realizează pe baza unei mulţimi de antrenare ce cuprinde exemple de antrenare pozitive (ferestre ce conţin feţe) şi exemple de antrenare negative (ferestre ce nu conţin feţe). Pentru acest proiect veţi lucra numai cu ferestre de dimensiuni 36×36 pixeli. Câteva exemple pozitive şi negative sunt ilustrate în Figura 2. Imaginile ce conţin exemplele pozitive au dimensiunile 36×36 pixeli. Imaginile ce conţin exemple negative sunt de diverse dimensiuni. Este important de remarcat următorul aspect: într-o imagine negativă ce nu conţine feţe orice fereastră de dimensiuni 36×36 pixeli reprezintă un exemplu negativ de antrenare.

Toate exemplele de antrenare vor fi prelucrate la nivel de imagini grayscale (tonuri de gri) şi nu de imagini RGB (cu 3 canale). Clasificatorul va fi antrenat astfel încât să decidă pe

Figura 2: **Exemple de antrenare. Rândul de sus**: exemple pozitive de antrenare (de dimensiuni 36×36 pixeli). **Rândul de jos**: imagini negative de antrenare. Fiecare fereastră de dimensiuni 36×36 pixeli din imagine reprezintă un exemplu negativ de antrenare.

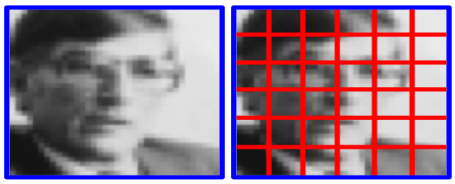

Figura 3: **Descrierea conţinutului vizual al unei imagini cu HOG.** (a) Imagine iniţială de dimensuni 36 $\times$ 36 pixeli. (b) Imagine împărţită în celule de dimensiune 6 $\times$ 6 pixeli.

baza conţinutului vizual al unei ferestre grayscale dacă aceasta reprezintă sau nu o faţă.

Histograme de gradienţi orientaţi. Veţi descrie conţinutul vizual al unei ferestre printr-un descriptor local, folosind histograme de gradienţi orientaţi (Figura 3). În acest proiect veţi lucra cu un descriptor gata implementat. Descriptorul este implementat de funcţia *hog* din biblioteca Sklearn şi are forma următoare:

descriptors = hog(img, pixels_per_cell=(dim_hog_cell, dim_hog_cell), cells_per_block=(2, 2), feature_vector=False)

Funcţia primeşte ca input o imagine, o împarte în celule pătratice de dimensiune *dim_hog_cell* (al doilea argument al funcţiei) şi descrie conţinutul fiecărei celule printr-un descriptor de dimensiune 36. Găsiţi descrierea funcţiei *hog* aici https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.hog. Pentru exemplele pozitive de dimensiune 36 $\times$ 36 şi *dim_hog_cell* = 6 pixeli (Figura 3), descriptorul HOG corespunzător va avea dimensiunile 5 $\times$ 5 $\times$ 36 (sunt 5 blocuri pe verticală şi 5 blocuri pe orizontală, fiecare conţinând 2 $\times$ 2 celule) . Pentru o imagine de dimensiune 100 $\times$ 100 şi *dim_hog_cell* = 6 pixeli, descriptorul HOG corespunzător va avea dimensiunile 15 $\times$ 15 $\times$ 36.

Descriptori pentru exemple pozitive şi negative. Antrenarea unui clasificator se realizează de către funcţia *train* din clasa *FacialDetector* pe baza descriptorilor pentru exemple pozitive şi exemple negative. Obţineţi descriptorii pentru exemple pozitive şi pentru exemplele negative apelând funcţiile *get_positive_descriptors* respectiv *get_negative_descriptors*. Funcţia *train* este scrisă în întregime, în timp ce funcţiile *get_positive_descriptors* şi *get_positive_descriptors* trebuiesc completate. Funcţia *get_positive_descriptors* prelucrează imagini de antrenare pozitive de dimensiuni 36 $\times$ 36 de pixeli din directorul **../data/exemplePozitive/** ce conţine 6713 imagini cu feţe centrate (Figura 2, rândul de sus).

Valorile parametrilor din clasa *Parameters* sunt setate astfel: *dim_window* = 36 de pixeli, *dim_hog_cell* = 6 pixeli, *dim_descriptor_cell* = 36. Astfel, descriptorul unui exemplu pozitiv va avea dimensiunea (36 / 6 - 1) * (36 / 6 - 1) * 36 = 900 (5 blocuri pe verticală $\times$ 5 blocuri pe orizontală $\times$ lungime descriptor bloc, fiecare bloc conţine 2$\times$2 celule). Prin urmare dimensiunile matricei *positive_descriptors* vor fi 6713 $\times$ 900.

Funcţia *get_negative_descriptors* prelucrează imagini de antrenare negative de diverse dimensiuni din directorul **../data/exempleNegative/** ce conţine 274 imagini negative (Fi-

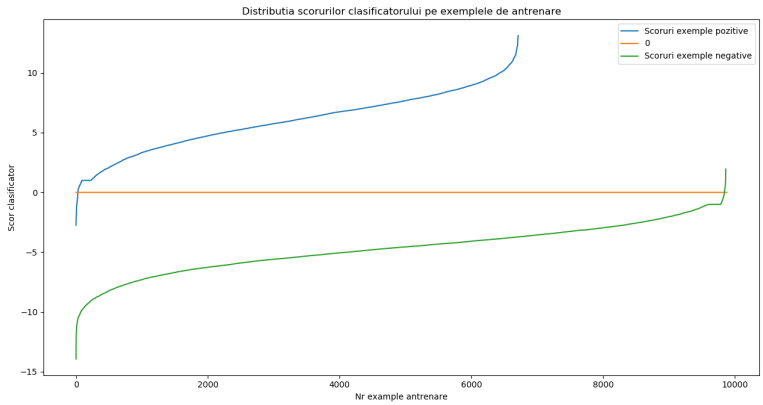

Figura 4: **Vizualizarea scorurilor asignate de clasificatorul învăţat exemplelor de antrenare.** Cu verde: scorurile exemplelor negative. Cu albastru: scorurile exemplelor pozitive. Un clasificator perfect ar asigna scoruri > 0 pentru toate exemplele pozitive şi scoruri < 0 pentru toate exemplele negative.

gura 2, rândul de jos).

Spre deosebire de funcţia precedentă, în această funcţie trebuie să generaţi pentru fiecare imagine în parte ferestre aleatoare iar apoi calculaţi descriptorii (de dimensiune 900) pentru aceste ferestre. Numărul total al ferestrelor generate aleator din cele 274 de imagini negative este setat în cod ca fiind egal cu 10000. Prin urmare dimensiunile matricei *get_negative_descriptors* vor fi 10000×900.

Clasificator liniar. Funcţia *train* învaţă un clasificator liniar sub forma unui vector $\mathbf{w}$ de dimensiune egală cu dimensiunea descriptorului unei ferestre ($= 900$ pentru parametri iniţiali) şi a unui scalar b astfel încât scorul clasificatorului liniar $(\mathbf{w}, b)$ pentru o fereastră f cu descriptorul asociat $\mathbf{d_f}$ se poate calcula astfel:

$$scor(f) =< \mathbf{w}, \mathbf{d_f} > +b.$$

Ideal, clasificatorul liniar separă perfect descriptorii exemplelor de antrenare pozitive şi negative (Figura 4) asignând scoruri pozitive pentru exemplele pozitive şi scoruri negative pentru exemplele negative. Clasificatorul liniar perfect ar satisface relaţia:

$$scor(f_p) =< \mathbf{w}, \mathbf{d_{f_p}} > +b > 0$$

pentru ferestrele f_p conţinând exemple pozitive şi relaţia:

$$scor(f_n) =< \mathbf{w}, \mathbf{d_{f_n}} > +b < 0$$

pentru ferestrele f_n conţinând exemple negative. În practică, este foarte posibil ca cele două mulţimi de descriptori să nu fie liniar separabile (să nu existe nici un hiperplan dat de $(\mathbf{w}, b)$ cu proprietatea de mai sus). Astfel putem întâlni situaţia când exemplele

pozitive primesc un scor negativ din partea clasificatorului (exemple fals negative) sau când exemplele negative primesc un scor pozitiv din partea clasificatorului (exemple fals pozitive). Figura 4 arată scorurile asignate de clasificator exemplelor de antrenare.

Antrenarea cu exemple puternic negative. O modalitate (opţională) de a îmbunătăţi performanţa clasificatorului liniar este de a rula detectorul facial (vedeţi funcţia *run* din secţiunea următoare) pe imaginile de antrenare negative şi a considera toate detecţiile cu scor > 0 ca exemple adiţionale negative (aceste exemple se numesc exemple puternic negative). Toate ferestrele din imaginile de antrenare negative ar trebui să primească din partea clasificatorului un scor negativ. Ferestrele care nu conţin feţe dar au un scor > 0 vor fi asimilate cu detecţii fals pozitive.

1.2 Etapa de testare (5 puncte)

Pentru o imagine test, detectarea feţelor se realizează prin glisarea unei ferestre de diferite mărimi de la stânga la dreapta şi de sus în jos în imagine şi clasificarea fiecărei fereastre pe baza clasificatorului învăţat în etapa anterioară. Ferestrele din imagine care au un scor > 0 şi sunt maxime locale (nu există altă detecţie de scor mai mare care se suprapune cu ea) localizează feţele. O variantă similară cu glisarea unei ferestre de diferite mărimi într-o imagine este glisarea unei ferestre de dimensiuni constante (36×36 pixeli) în imagine şi redimensionarea acesteia. Funcţia *run* implementează o astfel de soluţie.

Variabila *detections* este o matrice (numpy array) care conţine pe fiecare linie în formatul [xmin, ymin, xmax, ymax] coordonatele ferestrei care localizează o faţă. Variabila *scores* reţine scorul fiecărei detecţii. Variabila $file\_names$ reţine numele imaginii procesate (este nevoie de acest nume în protocolul de evaluare al performantei detectorului facial programat de voi).

Glisarea unei ferestre. Pentru o imagine de dimensiuni $L \times C$ pixeli obţineţi mai întâi descriptorul HOG asociat imaginii de dimensiuni $l * c * 36$ folosind funcţia *hog*.

Apoi pentru cele l celule orizontale şi c celule verticale din imagine, glisaţi o fereastră care conţine $k \times k$ celule ($k = dim\_window/dim\_hog\_cell - 1$). În total vor fi $(l - k + 1) * (c - k + 1)$ ferestre de clasificat. Repetaţi algoritmul redimensionând imaginea pentru a obţine un detector care poate localiza feţe de diverse mărimi (Figura 1).

Localizarea corectă a feţelor în imaginile test. Perfomanţa unui detector facial se măsoară prin capacitatea lui de localiza corect feţele în imagini. O faţă este localizată corect dacă detecţia returnată de detector se suprapune cu mai mult de 30% (cu fereastra dreptunghiulară adnotată în imaginea test). Suprapunerea dintre cele două ferestre f_1 şi f_2 se calculează pe baza coordonatelor lor folosind formula $suprapunere(f_1, f_2) = intersectie(f_1, f_2)/reuniune(f_1, f_2)$. Această formulă este calculată în interiorul funcţiei *non_maximum_suppression*.

Eliminarea non-maximelor. Protocolul de evaluare al performanţei detectorului facial penalizează detecţiile care se suprapun foarte mult. Numai o singură detecţie (cea care acoperă cel mai mult exemplul adnotat) este considerată corectă, celelalte vor fi

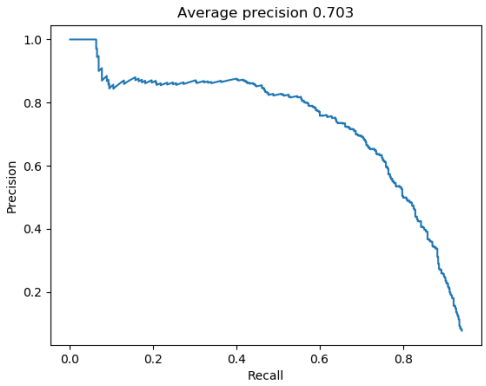

Figura 5: **Grafic precizie-recall**.

considerate detecţii fals pozitive. Pentru a obţine o perfomanţă mare a detectorului facial este necesar să folosiţi funcţia *non_maximum_suppression* în interiorul funcţiei *run*. Funcţia *non_maximal_suppression* va elimina toate detecţiile care se suprapun cu o altă detecţie de scor mai mare.

Seturi de date pentru testarea detectorului facial. Pentru testarea detectorului vostru facial veţi folosi setul de date $CMU + MIT$ ce conţine 130 de imagini cu 511 feţe adnotate. Acest set de date se găseşte în directorul **'../data/exempleTest/CMU+MIT/'**. De asemenea puteţi evalua calitativ performanţa detectorului vostru (nu există adnotări, setaţi Parameters.has_annotations = False) pe imaginile realizate la curs din directorul **../data/exempleTest/CursVA/**.

Protocolul de evaluare. Cuantificăm performanţa unui detector facial de a localiza corect feţe în imagini test prin două valori:

- *precizia*: procentul de detecţii returnate de detectorul facial ca fiind ferestre ce conţin feţe. În cazul ideal, detectorul vostru facial are o precizie = 1 = 100%, adică fiecare detecţie considerată de detector reprezintă o faţă.
- *recall* (=rată de detectare): procentul de feţe din imaginile test localizate corect. În cazul ideal detectorul vostru facial are un recall = 1 = 100%, adică localizează corect toate feţele din imagine.

Combinăm cele două valori (precizie + recall) într-un grafic de tip precizie-recall (Figura 5). Fiecare punct de pe acest grafic reprezintă precizia şi recall-ul detectorului facial obţinute pentru toate detecţiile (ordonate descrescător după scor) care depăşesc un anumit scor prag (threshold). Întregul grafic se cuantifică numeric prin *precizia medie* care reprezintă aria de dedesubtul graficului. Funcţia *eval_detections* realizează acest grafic şi calculează precizia medie. Figura 5 ilustrează graficulul precizie-recall şi precizia medie = 0.703 pentru setul de date $CMU + MIT$ şi un detector facial cu parametri iniţiali. Un detector facial perfect ar avea precizia medie $= 1$, cel mai din dreapta punct de pe graficul funcţiei

Figura 6: **Exemple de detecţii pentru imaginile test**. Primul rând: rezultate pentru imagini adnotate. Cu galben: feţele adnotate. Cu verde: detecţiile adevărat pozitive (localizează corect o faţă) returnate de către detectorul facial. Cu roşu: detecţiile fals pozitive (nu localizează corect o faţă) returnate de către detectorul facial. Al doilea rând: rezultate pentru imagini fără adnotare. Cu verde: detecţiile returnate de către detectorul facial.

aflâdu-se în acest caz în colţul din dreapta sus.

Vizualizarea rezultatelor. Puteţi vizualiza rezultatele detectorului vostru facial folosind funcţia *show_detections_with_ground_truth* pentru setul de date $CMU + MIT$ sau funcţia *show_detections_without_ground_truth* pentru imaginile realizate la curs. Aveţi câteva exemple în Figura 6.

Experimente. Odată ce aţi implementat funcţiile ce trebuiesc completate pentru rularea proiectului (acestea sunt în număr de 3: *get_positive_descriptors*, *get_negative_descriptors*, *run* şi opţional antrenarea cu exemple puternic negative) puteţi realiza experimente cu detectorul vostru facial. În experimentele voastre veţi încerca să găsiţi cei mai buni parametri astfel încât detectorul vostru să funcţioneze cât mai bine. Cuantificăm performanţa detectorului vostru facial cu precizia medie pe care o obţineţi pe setul de date $CMU + MIT$. Un detector facial foarte bine antrenat ar trebui să atingă o precizie medie de peste 0.90. În realizarea experimentelor voastre încercaţi diferite valori pentru următorii parametri:

- dim_hog_cell (= 6 pixeli implicit);
- threshold (= 0 implicit);
- mărirea numărului de exemple pozitive (= 6713 implicit) şi negative (=10000 implicit);
- antrenarea cu exemple puternic negative.

Foarte important: valoarea $Parameters.overlap = 0.3$ care fixează pragul de 0.3 pentru ca o detecţie sa fie considerată corectă nu trebuie schimbată.

1.3 Predarea proiectului

Veţi fi notaţi cu 4 puncte pentru realizarea etapei de antrenare, cu 3 puncte pentru implementarea glisării ferestrei a detectorului facial şi cu alte 2 puncte pentru experimentele pe care le veţi face.

Puneţi într-o arhivă cu numele *tema5_cod+parametri.zip* codul vostru Python (fără imaginile din *data*), obiectul *params* (de tipul Parameters) şi modelul SVM. Puneţi într-un document cu numele *tema5_rezultate+vizualizari.pdf* următoarele:

(a) rezultatele diferitelor voastre experimente (minim 3 experimente ataşând pentru fiecare experiment graficul cu precizia medie - vedeţi Figura 5) precizând de fiecare dată valorile parametrilor folosiţi;

(b) rezultatul cel mai bun (graficul cu precizie medie) obţinut comentând valorile parametrilor;

(c) vizualizări obţinute pentru minim 5 imagini din setul de date $CMU + MIT$ şi minim 2 imagini realizate la curs;

Trimiteţi cele două fişiere (*tema5_cod+parametri.zip* şi *tema5_rezultate+vizualizari.pdf*) la adresa de email **bogdan.alexe@fmi.unibuc.ro** precizând numele şi grupa din care faceţi parte.
Termenul limită de predare a proiectului este marţi, 22 decembrie 2020, ora 23:59. Fiecare zi de întârziere în predarea temei se penalizează cu 1 punct în minus. Temele primite după data de 25 decembrie nu vor fi luate în calcul.